Colombian School of Engineering
Enterprise Architecture, 2020-2

MODULARIZATION WITH DOCKER, USING SPARK, LB AND MONGODB

Sergio Alejandro Bohórquez Alzate

22/09/2020

Contents

1 Introduction

In this document we will find a solution to the laboratory number 5 of enterprise architectures in which we will apply the concepts of virtualization and in turn modularization seen in class, the goal is to use docker to create containers that store each node of our architecture in isolation, these nodes will communicate and make the distribution of the load obtained by a client or webbootstrap. Finally we will make use of an AWS EC2 instance to deploy the system in the cloud.

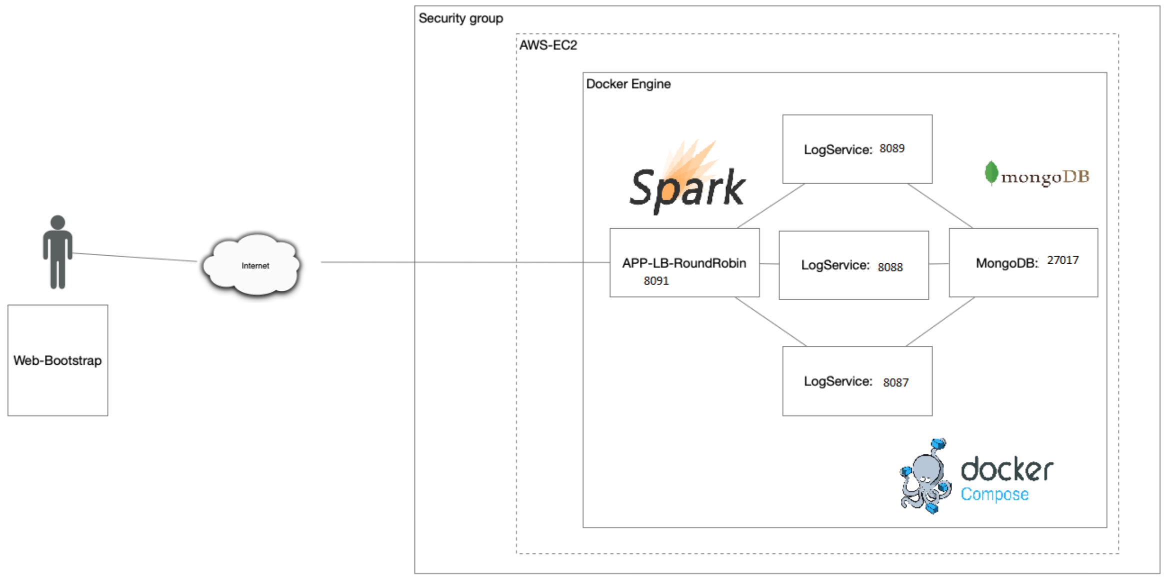
2 Objectives

- Understanding load balancers and their utility in a distributed system.
- Understanding docker virtualization and its utilities
- Learn about the operation of AWS's EC2

3 Architecture

3.1 Main architecture

In the following image we observe the main architecture of our system, also we will find an explanation of each one of the components. The architecture was defined in class.



- **Docker Engine** The docker engine is in charge of maintaining the docker images, these images are application instances and have all the infrastructure so they can run properly for example both the log service and the load balancer have a small operating system and the virtual machine of java virtualized within it where our application will run. From the images it is possible to take out several instances called containers these containers can be replicated from a single instance to ensure redundancy and availability.
  - MongoDB The component mongo db is responsible for providing a non-relational database instance in which you will have a collection of messages with date.
  - LogService The log service is in charge of providing two services: to obtain the last messages and to insert messages in the database. There will be three instances of it to ensure availability and redundancy in case any container fails and spark is used for implementation.

  - App-LB-RoundRobin The APP-LB component is responsible for providing a web client and a load balancer that distributes the requests with round robin and sends them to the server who gets the response.

- **AWS-EC2** AWS provides an instance of the necessary infrastructure in the cloud, we use a linux machine in which we will use our images uploaded to the docker hub and a docker compose that will build the necessary containers.

- **Security Group** AWS offers a basic security group called lswizard-2 in this security group we define 2 entry rules one for the telnet or ssh connection on port 22 and one for exposing our web client to the public on port 8091, the other ports (8087-8089) will be internal to the server and do not need to be accessed by users.

- **Web-Bootstrap** Internet users will be able to access our application through the client's facade exposed on the Internet with the rules defined above.

3.2 Definitions

- **Load Balancer** A load balancer is basically a hardware or software device that is placed in front of a set of servers serving an application and, as its name implies, assigns or balances the requests coming from the clients to the servers using some algorithm (from a simple round-robin to more sophisticated algorithms).[2]

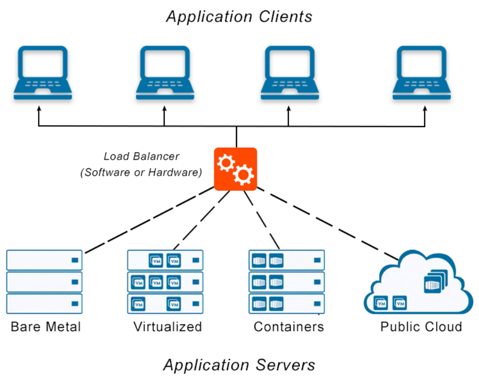


Figure 1: Load balacner Taked from avitnetworks.

A good load balancer should

  - minimize response times.
  - Improve service performance.
  - Avoid saturation.

- **Redundancy** Redundant systems, in software and enterprise architecture, are those in which critical data or hardware is repeated and that we want to ensure against possible failures that may arise from their continued use. In addition, we seek to increase the computing capacity and therefore the number of requests and availability.

- **AWS EC2** Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable cloud computing capabilities. It is designed to simplify the use of web-scale cloud computing for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides complete control over computing resources and allows you to run in the proven Amazon computing environment.[1]

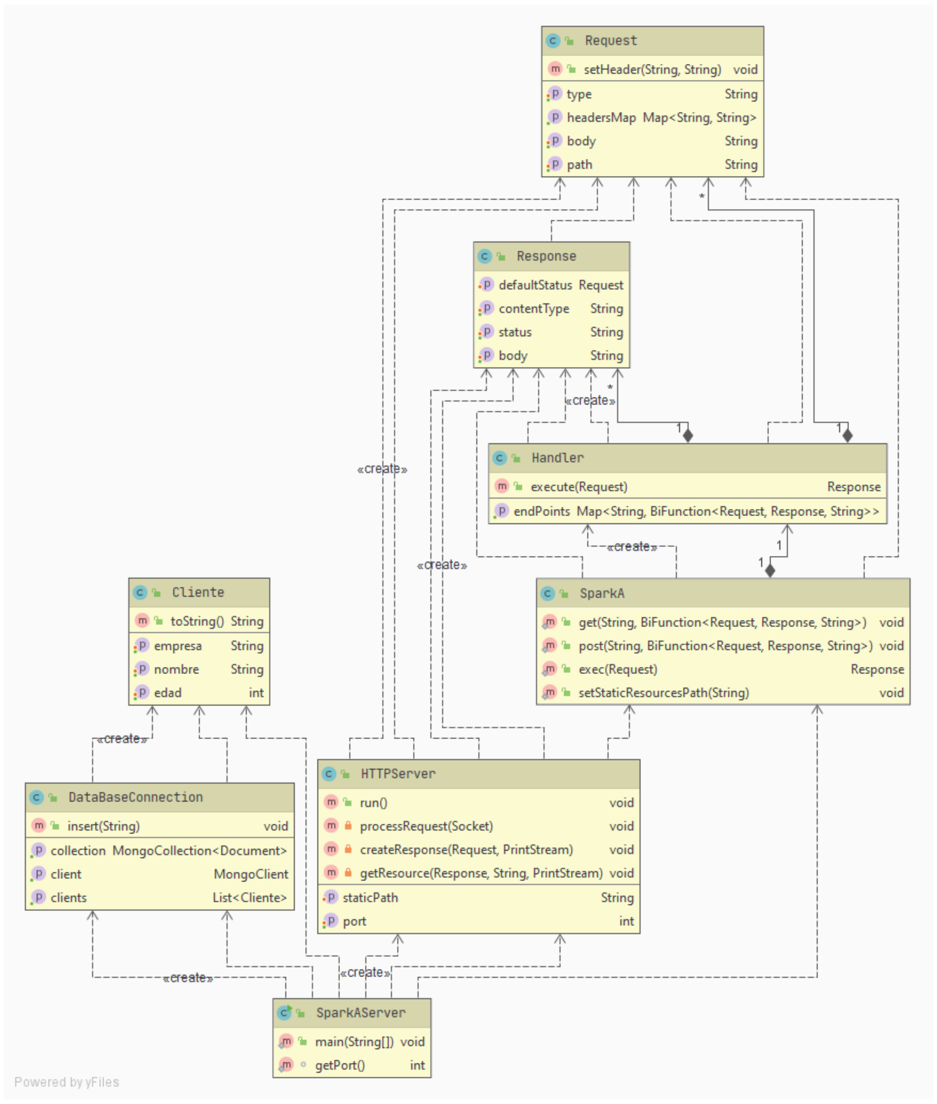
- **Docker** The idea behind Docker is to create lightweight, portable containers for software applications that can run on any machine with Docker installed, regardless of the operating system underneath, thus facilitating deployments as well.[3]

Figure 2: Docker logo Taked from docker.com .

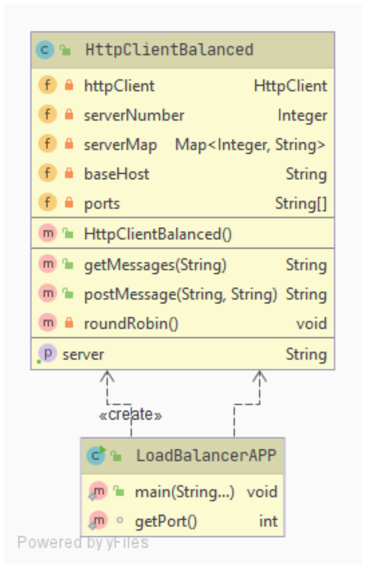
- **Distributed System** A distributed system is a set of computers that work together in a coordinated way, through the exchange of messages on the network, to achieve a goal. In such a system, status and programs are stored on multiple computers. Although the processes that take place are separated between the different participants, to the user it seems that he is working with only one computer.

3.3 Class Diagrams

- **Server** In the server class diagram we find important information about the architectural decisions made for the connection of the MongoDb database and the implementation of the REST service with spark.


Request
setHeader(String, String) void
type String
headersMap Map<String, String>
body String
path String
Response
defaultStatus Request
contentType String
status String
body String
Handler
execute(Request) Response
endPoints Map<String, BiFunction<Request, Response, String>>
SparkA
get(String, BiFunction<Request, Response, String>) void
post(String, BiFunction<Request, Response, String>) void
exec(Request) Response
setStaticResourcesPath(String) void
Cliente
toString() String
empresa String
nombre String
edad int
DataBaseConnection
insert(String) void
collection MongoCollection<Document>
client MongoClient
clients List<Cliente>
HTTPServer
run() void
processRequest(Socket) void
createResponse(Request, PrintStream) void
getResource(Response, String, PrintStream) void
staticPath String
port int
SparkAServer
main(String[]) void
getPort() int
«create»
1
*
Powered by yFiles

- **Client** In the client class diagram we find important information about the client implementation, as we can see the load balancer uses an http client with round robin.



4 Testing the System

4.1 Technology Stack

- Java Development Kit
- Maven
- Docker
- AWS EC2
- Spark
- MongoDB

5 Conclusions

- Load balancers allow communication between distributed systems to have more processing availability and better latency.
- Cloud computing and virtualization services enable service-oriented architecture to be more efficient.
- Virtualization allows us to reduce the amount of system failures and package applications as a whole including the OS.

References

[1] *Amazon EC2*. `https://aws.amazon.com/es/ec2/`. Accessed on 2020-08-22.

[2] *Balanceadores de carga*. `https://es.wikipedia.org/wiki/Balance_de_carga`. Accessed on 2020-08-22.

[3] *Docker*. `https://www.docker.com/resources/what-container`. Accessed on 2020-08-22.